Ruby trunk - Feature #11815

Proposal for method `Array#difference`

12/14/2015 08:27 AM - CaryInVictoria (Cary Swoveland)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

I propose that a method Array#difference be added to the Ruby core. *(Array#difference, which appears to be an alias of Array#-, was added to Ruby 2.6. Accordingly, I would propose that this method be named remove, but I will not make any changes here.)* It is similar to Array#- but for each element of the (array) argument it removes only one matching element from the receiver. For example:

```
a = [1,2,3,4,3,2,2,4]
b = [2,3,4,4,4]

a - b #=> [1]
c = a.difference b #=> [1, 3, 2, 2]
```

As you see, a contains three 2's and b contains 1, so the first 2 in a has been removed from a in constructing c. When b contains as least as many instances of an element as does a, c contains no instances of that element.

It could be implemented as follows:

```
class Array
  def difference(other)
    h = other.each_with_object(Hash.new(0)) { |e,h| h[e] += 1 }
    reject { |e| h[e] > 0 && h[e] -= 1 }
  end
end
```

Here are a few examples of its use:

Determine if two arrays of the same size contain the same elements

```
  a = [2,1,4,2,5,3,3,1]
  b = [3,4,1,1,2,3,5,2]
  a.difference(b).empty?
    #=> true
```

Identify an array's unique elements

```
  a = [1,3,2,4,3,4]
  u = a.uniq #=> [1, 2, 3, 4]
  u - a.difference(u) #=> [1, 2]
```

Identify a maximal number of 1-1 matches between the elements of two arrays and return an array of all elements from both arrays that were not matched

```
  a = [1, 2, 4, 2, 1, 7, 4, 2, 9]
  b = [4, 7, 3, 2, 2, 7]
  a.difference(b).concat(b.difference(a))
    #=> [1, 1, 4, 2, 9, 3, 7]
```

To remove elements from a starting at the end (rather the beginning) of a:

```
a = [1,2,3,4,3,2,2,4]
b = [2,3,4,4,4]

a.reverse.difference(b).reverse #=> [1,2,3,2]
```

Array#difference! could be defined in the obvious way.

More information is in my answer to this SO question.

History

#1 - 12/15/2015 04:33 AM - matz (Yukihiro Matsumoto)

Is there any real world example?

Matz.

#2 - 12/15/2015 05:15 AM - CaryInVictoria (Cary Swoveland)

Matz, alas, I cannot offer one. You see, Ruby--coding generally--is just a hobby for me. I spend a fair bit of time answering Ruby questions on SO and would have reached for this method on many occasions had it been available. Perhaps readers with development experience (everybody but me?) could reflect on whether this method would have been useful in projects they've worked on.

#3 - 12/15/2015 06:56 AM - CaryInVictoria (Cary Swoveland)

- Description updated

#4 - 12/15/2015 07:02 AM - CaryInVictoria (Cary Swoveland)

- Description updated

#5 - 12/15/2015 08:01 AM - duerst (Martin Dürst)

Cary Swoveland wrote:

> I spend a fair bit of time answering Ruby questions on SO and would have reached for this method on many occasions had it been available.

Then why don't you just provide pointers to those SO (StackOverflow?) questions, with explanations on how Array#difference would make things easier?

#6 - 12/15/2015 08:47 AM - CaryInVictoria (Cary Swoveland)

Martin Dürst wrote:

> Then why don't you just provide pointers to those SO (StackOverflow?) questions, with explanations on how Array#difference would make things easier?

Martin, see my SO answer here, which contains links to a number of questions where I did use the method in my answer.

#7 - 12/15/2015 12:27 PM - danielpclark (Daniel P. Clark)

I like how your **Array#difference** method works well with duplicate entries. I've only come across times where the difference of id references between two lists needed to be determined. In my case it's

```
a = [2, 4, 6, 8, 2, 4, 6, 8]
b = [1, 2, 3, 4, 1, 2, 3, 4]

# example
b - a
# => [1, 3, 1, 3]

b - a | a - b
# => [1, 3, 6, 8]
```

Like the example you first gave with added | b - a for getting two way evaluation on uniqueness. If I wanted to get the same thing with Array#difference it looks the same as my example above.

```
a = [2, 4, 6, 8, 2, 4, 6, 8]
b = [1, 2, 3, 4, 1, 2, 3, 4]

# example
b.difference(a)
# => [1, 3, 1, 3]

a.difference(b) | b.difference(a)
# => [1, 3, 6, 8]
```

So as to not cause confusion these are not the same as I will demonstrate with Cary Swoveland's input.

```
a = [1,2,3,4,3,2,2,4]
```

```
b = [2,3,4,4,4]

b.difference(a)
# => [4]
b - a
# => []

a.difference(b)
# => [1, 3, 2, 2]
a - b
# => [1]
```

As far as a real world use case for **Array#difference**: Service (A) exports all data to CSV files with a background worker. Service (B) exports to a database with a background worker. Sometimes a background worker crashes. Now to figure out what's missing we compare the difference between to two datasets. *One flaw in my example is there is no determination in the position the new data needs to be entered to match the other. In this case we would need to use something like Enumerator#with_index*

@Cary Swoveland; If I could make one recommendation on the implementation. I think it would be best to have it implemented as an **Enumerator** so it can be performed with lazy evaluation. That way when the difference is being compared we can perform operations along the way and "potentially" save system resources.

Here's the Enumerator that will give the same results as your code.

```
class Array
  def difference(other)
    cpy = other.dup
    Enumerator.new do |y|
      self.each do |e|
        ndx = cpy.index(e)
        if ndx
          cpy.delete_at(ndx)
        elsif !cpy.empty?
          y << e
        end
      end
    end
  end
end
```

I must admit that I am biased toward Enumerators over simple Array results. If this is never used on anything complex then it won't need to be an Enumerator.

#8 - 12/15/2015 05:37 PM - CaryInVictoria (Cary Swoveland)

Daniel, thank you for your interesting observations. I too am fond of enumerators ([true, false].cycle being a fav), and I can see the advantages here. In my second example, for example, one could write w1.chars.difference(w2.chars).none? #=> true, avoiding the need for the construction of a temporary array. On the other hand, if the method were often used in conjunction with other Array methods, tacking on to_a may become tiresome. The last line of my first example would become u - a.difference(u).to_a, my third example would be a.difference(b).to_a.concat(b.difference(a).to_a) and your example would be a.difference(b).to_a | b.difference(a).to_a.

#9 - 12/15/2015 05:52 PM - danielpclark (Daniel P. Clark)

I see your point. Looking into how Enumerator::Lazy works it looks like a good solution for having both.

```
class Array
  def difference(other)
    dup.tap do |cpy|
      other.each do |e|
        ndx = cpy.index(e)
        cpy.delete_at(ndx) if ndx
      end
    end
  end
end


class Enumerator::Lazy
  def difference(other)
    cpy = other.dup
    Lazy.new(self) do |y, e|
      ndx = cpy.index(e)
      if ndx
        cpy.delete_at(ndx)
      elsif !cpy.empty?
        y << e
      end
    end
  end
end

a = [1,2,3,4,3,2,2,4]
b = [2,3,4,4,4]

a.difference(b)
# => [1, 3, 2, 2]
a.lazy.difference(b).next
# => 1
a.lazy.difference(b).force
# => [1, 3, 2, 2]
```

This works well.

#10 - 12/18/2015 03:48 AM - rp.beltran (Ryan Beltran)

Yukihiro Matsumoto wrote:

> Is there any real world example?
>
> Matz.

I think I have a pretty good example. I'm implementing a function in Ruby that finds triples in an array for use in a pokerbot (recognizes if a hand is a triple). I already defined a function to check for doubles (which is relatively trivial to implement by comparing the original array to array.uniq), but triples are a little bit harder. There are several ways I could implement checking for triples, but a concise and efficient option would be to simply call:

getDoubles(array.difference(array.uniq))

This works because if an array has a triple, then the difference between it and a set of it's unique values will have a double of the same value as the triple. This is much nicer than any methods I have come up with iteratively.

With array. difference can also define a more elegant alternative to what I was using for doubles as:

array.difference(array.uniq).uniq

If I want to build the lists of doubles, triples, and four of a kind, I can define them even more elegantly as:

```
doubles = array.difference(array.uniq)
triples = doubles.difference(doubles.uniq)
fours   = triples.difference(triples.uniq)
```

And this pattern continues on as shown.

#11 - 12/18/2015 10:17 AM - duerst (Martin Dürst)

Ryan Beltran wrote:

> I think I have a pretty good example. I'm implementing a function in Ruby that finds triples in an array for use in a pokerbot (recognizes if a hand is a triple). I already defined a function to check for doubles (which is relatively trivial to implement by comparing the original array to array.uniq), but triples are a little bit harder. There are several ways I could implement checking for triples, but a concise and efficient option would be to simply call:
>
> getDoubles(array.difference(array.uniq))
>
> ```
> doubles = array.difference(array.uniq)
> triples = doubles.difference(doubles)
> fours   = triples.difference(triples)
> ```

An even more straightforward way is to use group_by:

```
n_tuples = array.group_by {|e| e}.values.group_by(&:length)

doubles = n_tuples[2]
triples = n_tuples[3]
fours   = n_tuples[4]
# and so on
~~~ ruby
```

```
We need group_by two times because the first one groups items with the same value, and the second organizes
these by numbers. As an example, if we start with [1,2,2,3,3,3,4,4,4,4,5,5,5,7,8,8] then after the first
```

```
group_by, we have
{1=>[1], 2=>[2, 2], 3=>[3, 3, 3], 4=>[4, 4, 4, 4], 5=>[5, 5, 5], 7=>[7], 8=>[8, 8]}.
Of this, we only need the values: [[1], [2, 2], [3, 3, 3], [4, 4, 4, 4], [5, 5, 5], [7], [8, 8]].
Then we group by length and get:
{1=>[[1], [7]], 2=>[[2, 2], [8, 8]], 3=>[[3, 3, 3], [5, 5, 5]], 4=>[[4, 4, 4, 4]]}
To get an unique value (i.e. 3 instead of [3, 3, 3]), just use .map(&:first).
```

#12 - 12/19/2015 07:22 PM - CaryInVictoria (Cary Swoveland)

- Description updated

#13 - 08/19/2016 10:43 PM - CaryInVictoria (Cary Swoveland)

- Description updated

Implemented the method in a clearer and more efficient manner.

#14 - 09/15/2016 04:43 PM - CaryInVictoria (Cary Swoveland)

- Description updated

#15 - 09/10/2017 11:56 PM - CaryInVictoria (Cary Swoveland)

- Description updated

#16 - 09/29/2018 02:14 PM - febeling (Florian Ebeling)

I could use this method for fixing this bug [1] in ActiveRecord.

To me it looks like a valuable addition.

It's about replacing collection associations which can be partially persisted, with persisted records being handled differently as an optimization.

1 https://github.com/rails/rails/issues/33942

#17 - 09/30/2018 12:16 PM - febeling (Florian Ebeling)

And analogously an method 'Array#intersection' would be valuable. Implementation could share most code.

#18 - 10/05/2018 08:04 PM - febeling (Florian Ebeling)

Now that Array#difference has in #14097 become an operator with set semantics, this proposal should probably be closed.

#19 - 12/21/2018 11:18 PM - CaryInVictoria (Cary Swoveland)

- Description updated